BUILDING DATABASE SYSTEMS

In the second in our series, Wilf Hey explains how relational structures provide both power and compactness within a database application.

In the first part of this series, we went through the steps necessary for building a record structure, made up of several fields. Our database file is intended to store details of magazine articles, with each record representing one article. Therefore fields such as magazine name, issue number, article title, and author are among those we want to build into each record.

The data supplied with the *SuperDisk* was limited to one issue of PC PLUS, but we planned a structure that would accommodate the addition of articles from other magazines as well. For this reason there is a field within the record structure called MAGAZINE.

At first, it may seem right to allow 25 characters for the MAGAZINE field, so that the full name can be recorded. Yet there are a few things we should consider. First, there is only a limited number of magazines you will wish to include within the database. Suppose you regularly read ten magazines; in this case, a shortened form for each magazine name could be developed. In

STARTING OFF

This series is set up as two parts, running in parallel over several months. In Designing a Database we look at the basic principles that apply to designing any database project – whatever database package or programming language you use. In *Developing with SR-Info* we apply these principles to a project started in issue 54 (March 91), using *SR-Info* – the language featured on the *SuperDisk* that month.

If you wish to follow this programming project, you will need to have installed *SR-Info* from the March *SuperDisk*.

DEVELOPING WITH SR-INFO

If you are unsure about how to start up in *SR-Info* please consult the explanation in the text given last month, in the first installment of our *Building Database Systems* series.

Last month, we created MAGFILE.DBF, a database file containing the full name and details of each of the magazines whose articles will be in the database. Initially we created it with only two fields in each record – MAGAZINE (three characters per field) and FULLNAME (twenty-five characters per field).

We didn't create any records for that file – nor did we create an index. Before we do that, let's modify the structure of MAGFILE so that each record contains fields for editor, address and frequency.

```
SELECT 1
USE MAGFILE
MODIFY
```

This will bring onto the screen a list of the fields in the structure currently, and allow you to modify that structure by adding, deleting or changing fields.

It is quite permissible to MODIFY a file, even if it has records in it. The major problem with this is that you would still have to BROWSE or EDIT through the records, if only to set up values for the new fields in each record. Just one more point to note; after you finish a MODIFY operation, you must USE the changed file again. (For safety, MODIFY finishes by declaring the current USE invalid).

INDEX LINKS

We can create an index for a file at any time, but it is common to do this at the same time we create the file itself – even before records are put into the database. Then, every time a new record is added, the index is kept up-to-date automatically. Again, you have total flexibility – you can rebuild an index any time you wish, whether records exist or not. (The more records that already exist, the longer it takes to organise the index).

Remember that you must include the names of every index to a file each time you USE that file (from last month, when you indexed the ARTICLES database). Refer to last month's article to remind yourself how INDEX ON creates an index, and SET INDEX TO includes all indexes along with the preceding USE command.

Now create an index MAGREF for the newly-modified MAGFILE. (This really needs to be done only once, if we remember to SET INDEX TO MAGREF every time we USE MAGFILE in the future).

```
USE MAGFILE
INDEX ON MAGAZINE TO MAGREF
```

Let's create a single magazine reference to MAGFILE, so that we can establish a relational link between ARTICLES and MAGFILE.

It's worth reviewing the convenient way *SR-Info* remembers the USE of up to ten files at a time – by preceding the first USE of a file with a SELECT command. After that, doing a SELECT to the same number sets the system to deal with the USE that last applied.

This is easier than it sounds; let's demonstrate with an experiment:

```
SELECT 1
USE MAGFILE
SET INDEX TO MAGREF
STATUS
```

our example, the MAGAZINE field is only three characters long – and PC PLUS is shortened to PCP in each record currently in the database. This saves 22 characters per record in the database (and much more in key storage), so more records can be stored in a given amount of disk space.

Of course PCP is not a desirable abbreviation to appear on screen or in a report, so you may wish to have a table in your programs, modifying PCP to PC PLUS (along with other abbreviations you have used). There are several disadvantages to this method. It requires copying the same code into several different programs. It also makes administration difficult when you (or any other user of the database) begin putting records into the database that refer to a new magazine – one that you could not have anticipated ever being in the database.

The good database solution is to create another database file, instead of a table of abbreviations. This file will consist of one record for each abbreviated magazine name. Each record must contain fields for the MAGAZINE (three characters long) and its corresponding FULLNAME (twenty-five characters long); it may be good to include ADDRESS, EDITOR and FREQUENCY fields, and any others relevant to that magazine. Any new magazine title to be included in the system can be added to the file without any change to the programs gathering information together for processing.

ESTABLISHING A RELATIONSHIP

It is this way of dealing with information – distributing it over more than one database file, and linking those files through abbreviated keys – that is called relational database programming.

There are usually several techniques for dealing with related files. However it is done, effectively you are gathering together all the available information relating to a particular record. For example, the short name PCP in the ARTICLES database file provides a relational link to all the data about PCP – its full name, address, editor and so on. You can build a relational database application with more than one link; on a coming *SuperDisk* you will find SUBJFILE.DBF which is intended for linking to ARTICLES by sharing the SUBJECT abbreviation.

In this month's *Developing with SR-Info* below, a simple statement links related records in different files. When a record from the primary database is called into memory (however this is done), all the supportive databases are aligned with that record. All the relevant information is in memory simultaneously – wherever it was stored.

We shall see in a future part of this series how we can take the information from several related files that are aligned in this fashion, and put together reports that do not have to suffer shortened forms or incomplete details. Even though the address of the magazine does not appear with each item of the ARTICLES database, the correct one can (if desired) be printed alongside the details of each article.

Many reports boil down to good database design, a few relational linkage statements, and a handful of statements picking the desired fields from their records, already pulled into memory.

BEST OF BOTH WORLDS

When database development languages were first being designed, new emphasis was placed on the best program design concepts formulated up to that time. The idea of the database file 'knowing' its own structure is one of these ideas. Programs using a database need to know virtually nothing about the records in the database, and very little about the fields within each record. (In fact, it usually has to know nothing but the name of fields – and then only of the fields it needs). It is for this reason that even quite drastic changes to a database will

The STATUS command will identify – for the first six (of ten) SELECT slots – what database files and indexes are being remembered:

```
SELECT 2
USE ARTICLES
SET INDEX TO AUTHOR, SUBJECT, TITLE
STATUS
```

Note the asterisk on the STATUS display, indicating which database is selected and active. Now check what happens when you key:

```
SELECT 1
STATUS
```

You will see that the `SELECT 1` has, in effect, meant that you do not have to `USE MAGFILE` again, nor `SET INDEX TO MAGREF`, because these facts are remembered. Now, with MAGFILE active, let's key a single record, with MAGAZINE field set to 'PCP' and FULLNAME set to 'PC PLUS'; do this with the APPEND command.

RELATIONAL DATABASES

SR-Info has instructions that enable you to link database files together, using key fields that appear in both files.

You can link databases together so, for example, when a record in ARTICLES.DBF is brought into memory, the matching MAGFILE.DBF record (with the MAGAZINE field also matching) is also brought into memory. This can only be done when the secondary file (MAGFILE) is indexed in the same order as that of the matching field.

You can then experiment with this in the following way; add a record to each of the two database files (with the SELECT and APPEND commands) using the same nonsensical MAGAZINE abbreviation for each – perhaps try VIZ. Now establish the relationship between the files in the following way:

```
SELECT 2
SET RELATION ON MAGAZINE TO 1
```

where `SELECT 2` points you to ARTICLES (the primary, or 'driving' database) and the figure '1' in the second command points to the slot in the supportive database l (that is MAGFILE). To test this feature out, try the following lines:

```
LOCATE ON MAGAZINE='VIZ'
? MAGAZINE
? FULLNAME£1
```

(This last instruction displays the record in slot 1, which we have been pointing to in the MAGFILE database). Now compare the result when you try this:

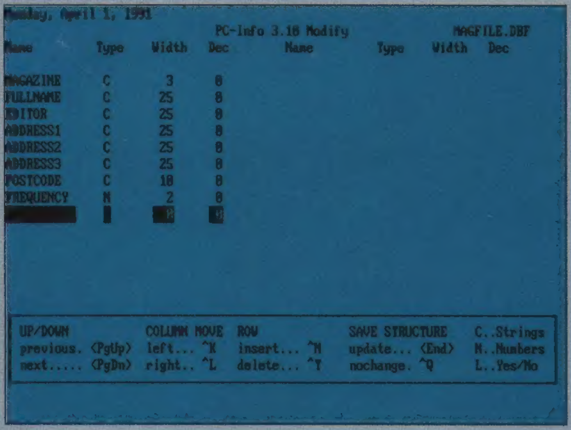
```
LOCATE ON MAGAZINE='PCP'
? MAGAZINE
? FULLNAME£1
```

The '?' command displays the specified field on screen – normally from the current active record. In the last instruction, since '#1' is added immediately after a field's name, *SR-Info* displays that field from slot 1 (pointing to the MAGFILE database). Now repeat that experiment after keying `SET RELATION OFF` first, and you will see that FULLNAME#1 doesn't change along with the current record (which changes with the LOCATE ON command).

WELL-EARNED REST

You have absorbed a lot during this session. Type `QUIT`, take a deep breath, and relax for a while!

MODIFY COMMAND



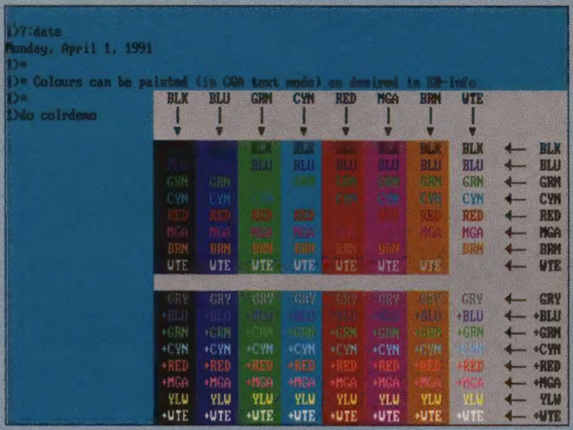
```
Monday, April 1, 1991
                         PC-Info 3.10 Modify                  MAGFILE.DBF
Name         Type   Width   Dec         Name         Type   Width   Dec

MAGAZINE      C       3      0
FULLNAME      C      25      0
EDITOR        C      25      0
ADDRESS1      C      25      0
ADDRESS2      C      25      0
ADDRESS3      C      25      0
POSTCODE      C      10      0
FREQUENCY     N       2      0


 UP/DOWN            COLUMN MOVE  ROW            SAVE STRUCTURE    C..Strings
 previous. <PgUp>   left... ^K   insert... ^N   update... <End>   N..Numbers
 next..... <PgDn>   right.. ^L   delete... ^Y   nochange. ^Q      L..Yes/No
```

● It's easy to add fields to a record structure; executing MODIFY brings to the screen the current record structure, and a summary of control keys at the bottom of the screen gives all available options.

PROGRAMMING STRUCTURES

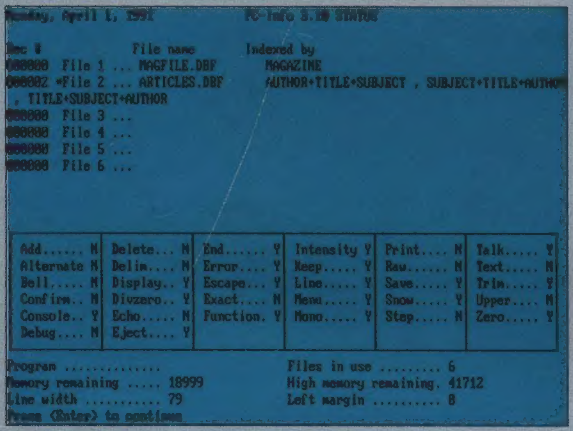


```
1>?:date
Monday, April 1, 1991
1>*
1>* Colours can be painted (in CGA text mode) as desired in SB-Info
1>*                BLK  BLU  GRN  CYN  RED  MGA  BRN  WTE
1>do colrdemo       |    |    |    |    |    |    |    |
                    ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼
                         BLK  BLK  BLK  BLK  BLK  BLK  BLK  ←  BLK
                    BLU       BLU  BLU  BLU  BLU  BLU  BLU  ←  BLU
                    GRN  GRN       GRN  GRN  GRN  GRN  GRN  ←  GRN
                    CYN  CYN  CYN       CYN  CYN  CYN  CYN  ←  CYN
                    RED  RED  RED  RED       RED  RED  RED  ←  RED
                    MGA  MGA  MGA  MGA  MGA       MGA  MGA  ←  MGA
                    BRN  BRN  BRN  BRN  BRN  BRN       BRN  ←  BRN
                    WTE  WTE  WTE  WTE  WTE  WTE  WTE       ←  WTE

                    GRY  GRY  GRY  GRY  GRY  GRY  GRY  GRY  ←  GRY
                   +BLU +BLU +BLU +BLU +BLU +BLU +BLU +BLU  ← +BLU
                   +GRN +GRN +GRN +GRN +GRN +GRN +GRN +GRN  ← +GRN
                   +CYN +CYN +CYN +CYN +CYN +CYN +CYN +CYN  ← +CYN
                   +RED +RED +RED +RED +RED +RED +RED +RED  ← +RED
                   +MGA +MGA +MGA +MGA +MGA +MGA +MGA +MGA  ← +MGA
                    YLW  YLW  YLW  YLW  YLW  YLW  YLW  YLW  ←  YLW
                   +WTE +WTE +WTE +WTE +WTE +WTE +WTE +WTE  ← +WTE
```

● The database programmer has complete control over the location and colour of screen displays in CGA text mode (eight background and sixteen foreground colours). This example demonstrates colour displays.

STATUS COMMAND



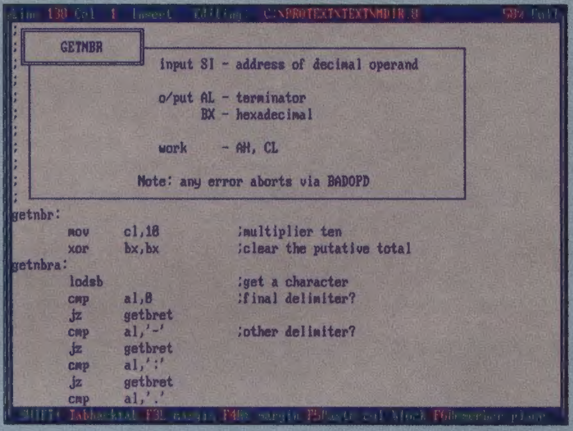
```
Monday, April 1, 1991              PC-Info 3.10 STATUS

Rec #              File name        Indexed by
000000  File 1 ... MAGFILE.DBF        MAGAZINE
000002 *File 2 ... ARTICLES.DBF       AUTHOR+TITLE+SUBJECT , SUBJECT+TITLE+AUTHO
 , TITLE+SUBJECT+AUTHOR
000000  File 3 ...
000000  File 4 ...
000000  File 5 ...
000000  File 6 ...


 Add...... N | Delete... N | End...... Y | Intensity Y | Print.... N | Talk..... Y
 Alternate N | Delim.... N | Error.... Y | Keep..... Y | Raw...... N | Text..... N
 Bell..... N | Display.. Y | Escape... Y | Line..... Y | Save..... Y | Trim..... Y
 Confirm.. N | Divzero.. Y | Exact.... N | Menu..... Y | Snow..... Y | Upper.... N
 Console.. Y | Echo..... N | Function. Y | Mono..... Y | Step..... N | Zero..... Y
 Debug.... N | Eject.... Y |             |             |             |
Program ..............              Files in use ......... 6
Memory remaining ..... 18999        High memory remaining. 41712
Line width ........... 79           Left margin .......... 0
Press <Enter> to continue
```

● Several database files and indexes are memorised, even though only one file is 'selected' at any one moment. The current files and indexes associated with 'select' slots are displayed by the STATUS command.

STRUCTURES



```
Line 130 Col 1  Insert  [illegible]  C:\PROTEXT\TEXT\MDIR.8                58% full
;
;  GETNBR
;                  input SI - address of decimal operand
;
;                  o/put AL - terminator
;                        BX - hexadecimal
;
;                  work     - AH, CL
;
;                Note: any error aborts via BADOPD
;
getnbr:
        mov     cl,10           ;multiplier ten
        xor     bx,bx           ;clear the putative total
getnbra:
        lodsb                   ;get a character
        cmp     al,0            ;final delimiter?
        jz      getbret
        cmp     al,'-'          ;other delimiter?
        jz      getbret
        cmp     al,':'
        jz      getbret
        cmp     al,'.'
SHIFT: Tabbackstab F3L margin F4R margin F5Paste col block F6[illegible]
```

● Within WRITE, you may use [ALT][F] to reorganise the entire program source code, so that related programming structures are indented clearly to the same extent. This makes programs more understandable.

very seldom affect the programs in the actual application.

Another design concept used throughout database languages is structured programming. Most early computer languages gave the programmer instructions to make a comparison or decision. Usually, the next instruction splits the program flow into two paths. Database languages are built with commands that make comparisons and decisions simple, yet keep a program's shape under control.

Some think it ironic that structured programming often makes for programs that are more compact and efficient, as well as more easily understood.

COMMAND STRUCTURES

The ARTILIST program from the March *SuperDisk* demonstrates many structures common to database programming:

1. **DO**
– in line 15 there is the command, DO ARTINEW. In fact, ARTINEW is a separate PRG – a source program written as a separate entity, but compiled together with ARTILIST to form one unit. ARTINEW.PRG could be developed and perfected independently, and then made available for inclusion with ARTILIST.PRG. In this way, even complex tasks can be treated as if they were each a single, one-line command.

ARTINEW.PRG was in fact written first – except for the WINDOW command in its third line. It can be compiled and executed separately. Its purpose is to build index files for the ARTICLES database.

As a contrast to this, LISTITEM.PRG (executed from line 19) is a task that hasn't been built yet! You'll get a chance to program it in a future issue. This technique allows us to build a project, test it, and use it before it is finished. Many parts of ARTILIST work, though the function '3. LIST CHOSEN ITEM' hasn't been programmed.

2. **DO WHILE .. ENDDO**
– from line 4 through to line 29 there is a special section of code; this is executed, and then repeated, until one of two terminating events occur.

One terminating event is that the condition specified ceases to be true; for example, DO WHILE .NOT. EOF() ensures that the instructions are executed continuously until the current database's end-of-file flag becomes set. The other terminating event is a BREAK being executed from somewhere within the group of instructions.

In the example in ARTILIST.PRG, the condition is '.T.' (which means TRUE), which can never cease to be true. There is a BREAK – in line 25 – which becomes executed when the function '6. EXIT PROGRAM' is selected.

This is very typical of code used for handling a group of menu items, where you wish to keep returning to the menu after each and every function until the

▶

Are you wasting your valuable time waiting for your printer to catch up?

Frontier's Printer-Q is the answer

Frontier Software's Printer-Q Expandable Printer Buffer

If your printer has an annoying habit of holding up your computer while it slowly prints out your latest creation then Frontier's Printer-Q expandable printer buffer is the inexpensive solution that you've been waiting for.

Easy To Use

The Printer-Q is a small device which sits between your computer and printer. Each time you print a document (whether it's a single page of simple text or multiple pages with large areas of graphics) the Printer-Q accepts the document from the computer at the fastest possible speed (up t[illegible]00 characters per second) and then feeds the printer at its quickest rate freeing up your expensive computer for another task almost immediately. You can even leave the program that just printed the document and use another totally different program. The Printer-Q doesn't mind.

The Printer-Q gives you the option to print up to eight copies of the same document - you simply print one copy from the computer and then press the repeat button on the Printer-Q for as many copies as you need and the Printer-Q takes care of the rest. While the copies are being printed the readout on the front of the Printer-Q tells you how many copies there are remaining to print.

Just Plug It In And Go

Installation is simply a matter of plugging your computer's printer cable into the back of the Printer-Q and then plugging the Printer-Q's cable into your printer. Turn on the Printer-Q and it's ready to go. There is no need to open either your printer or your computer.

Easily Expandable

The Printer-Q is easily expandable by you with no soldering or technical knowledge whatsoever. If you find out that the memory in most other buffers is not large enough for your requirements - the computer still has to wait for the printer because the buffer fills up too quickly - then you have to buy another larger printer buffer and scrap the current model. With the Printer-Q things couldn't be easier - starting at 128K (approximately forty pages of single spaced text) the Printer-Q is expandable up to a maximum 1MB in 128K stages. Each stage is an easy operation which takes minutes and costs very little - you simply open up the case, place another memory chip in a socket and close up the Printer-Q again.

Supplied Ready To Use

The Printer-Q is supplied with everything you need - A full manual, the amount of memory that you specify, 3 foot printer lead and power supply. The manual takes you through the easy installation and memory upgrade processes and includes a trouble shooting section if you have any problems.

Unlike most other printer buffers there is no need for any special installation software therefore the Printer-Q is compatible with nearly all computers from PCs through Amigas and STs to Amstrads. Since the Printer-Q is a separate peripheral it doesn't take any memory away from your computer which means that all of your programs can still run as normal. The Printer-Q supports the full Centronics standard, making it compatible with all Centronics printers and computers with Centronics ports.

Satisfaction Guaranteed

The Printer-Q is supplied under Frontier's ten day money back guarantee, which means that if you are unsatisfied with the Printer-Q then you can return it for a full refund within ten days of purchase. The Printer-Q carries a full twelve months' guarantee. Frontier manufacture the Printer-Q in the UK therefore knowledgeable support is always only a 'phone call away.

Prices

Printer-Q with 128K	£99.99
Printer-Q with 256K	£109.99
Printer-Q with 512K	£129.99
Printer-Q with 1MB	£169.99

(All models with less than 1MB are expandable by you with no soldering and no technical knowledge to 1MB).

Please add £3.00 for postage and packing to all orders under £150.00. Prices exclude VAT. Frontier accepts payment by Visa, Access, cheque or postal order. Prices subject to change without notice. Goods subject to availability.

Frontier Software

P.O. Box 113, Harrogate, North Yorkshire, England HG2 OBE. Phone (0423) 567140/530577. Fax (0423) 522874.

PROGRAMMER'S PREROGATIVE

You have great freedom with database design to change your mind at a future date. Not only can you easily add or change records within a database – you can, just as easily, add new fields to the record structure itself. In our example, the field FREQUENCY within MAGFILES may have been an afterthought, added to the structure several months after the database was first designed.

Other programming languages, not using database techniques, would require changes to all programs that refer to a file, once you decide to change its structure. But not so with a database! All the information about a database file is contained within the file itself; the programs rarely (if ever) need to be changed.

BUYING TIME

***SR-Info* is a product of Sub Rosa Inc of Downsview, Ontario, Canada, and is marketed in this country by New Star Software, on (0245) 265017.**

***SR-Info* is shareware. If you continue to make use of it after this series is over, or put it to use in a commercial application, you should register your copy with New Star.**

time the program is to be terminated.

3. **DO CASE .. ENDCASE**
– from line 13 through to line 26 there is a special section of code. This is executed only once, but does different things under different circumstances. It is actually a group of tiny portions of code (such as DO ADDITEM in line 23), preceded by the condition under which it should be executed (CASE MCHOICE=5 in line 22).

When the program is being executed, the command DO CASE determines which of the conditions prevails, and then executes the code immediately under that condition. Normally (but not always) a DO CASE structure will look like this one, with conditions that relate to the value of one variable or field, and each condition followed by very few lines of distinctive commands.

4. **PERFORM** (internal procedure)
– in line 2 there is a command to PERFORM NEWSEQ. After the main part of the program (which ends at line 34), you will find several groups of code, each headed with the word PROCEDURE and a distinctive name. The procedure called NEWSEQ runs from line 57 through to line 75. PERFORM will execute the procedure once, and then go on to the next command. It is very much like DO in its effect, except that the code for PERFORM is included within ARTILIST.PRG itself.

Be aware that all of these structures are whole units, but can have other structures within them; the TOTRECS procedure (lines 36 to 42) is quite able to PERFORM OLDSEQ, another procedure. Also note that the OLDSEQ procedure (lines 44 to 55) has a DO CASE .. ENDCASE structure embedded within it.

THE WRITE STUFF

When writing a program it's good practice to indent ceratin sections of the code to make it easier to follow the logic of the program. Within the WRITE command (which is used when you wish to create, inspect or change a program), you have the useful option to key [ALT][F].

This ignores whatever indentation exists, and provides its own, based on the above programming structures. For example, the DO CASE command will be aligned with its mate, the ENDCASE.

This particular editing feature is useful for another purpose – to make sure that each structure has both a beginning and an end. (If you forget that, ENDCASE indentation will not be correct, and the program listing will not end with its last statement beginning in column one.)

Even quite complicated routines can be converted to structured programming style quite easily. The most useful instructions for simulating unstructured routines from other computer languages are DO CASE – which can accomplish any one of many different things – and DO WHILE .T. (with a BREAK to get out of an eternal loop).

TRIAL RUN

Don't be afraid to code the odd empty PROCEDURE, or even whole program, (as in our example) because this method helps to define the overall relationship of all the related structures. It also permits you to write and test parts of a program even before you have sufficient time or information to complete other parts.

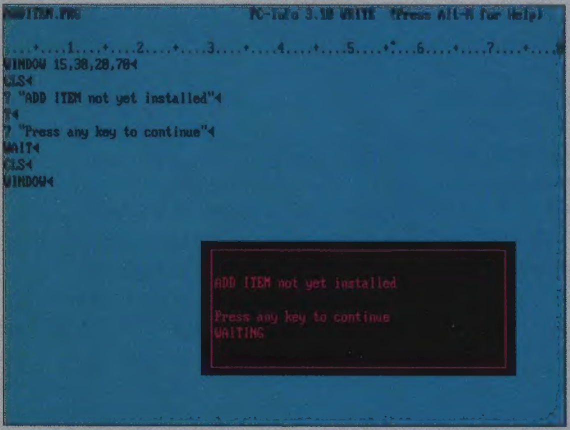
NEXT TIME

Next month we will be looking at how to design effective displays, and put together structures to create a working program.

It is now a good time to start considering what kind of enhancements you would like to see in the use of the Articles database; or perhaps you can think of a totally new database application that can be coded in *SR-Info*.

The best coded applications sent in to us after the completion of this series will earn a place on a future edition of the *SuperDisk*. ●

TRIAL RUN



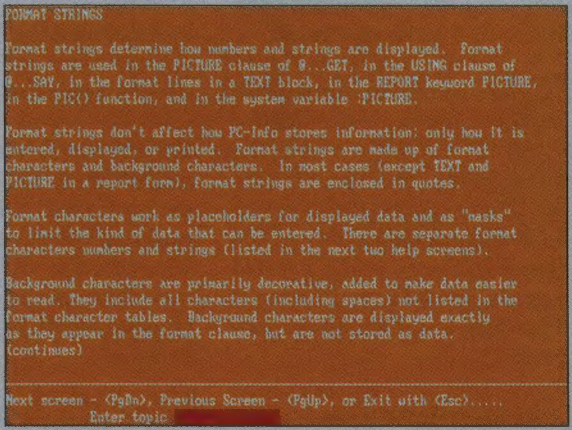
```
ADDITEM.PRG                          PC-Info 3.10 WRITE  (Press Alt-H for Help)
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
WINDOW 15,30,20,70◂
CLS◂
? "ADD ITEM not yet installed"◂
?◂
? "Press any key to continue"◂
WAIT◂
CLS◂
WINDOW◂

                        ADD ITEM not yet installed

                        Press any key to continue
                        WAITING
```

● It is often a useful technique to create a dummy program, which maps out the structure of the final program. The rest of the code can be added a section at a time later on.

HELP SYSTEM



```
FORMAT STRINGS

Format strings determine how numbers and strings are displayed.  Format
strings are used in the PICTURE clause of @...GET, in the USING clause of
@...SAY, in the format lines in a TEXT block, in the REPORT keyword PICTURE,
in the PIC() function, and in the system variable :PICTURE.

Format strings don't affect how PC-Info stores information: only how it is
entered, displayed, or printed.  Format strings are made up of format
characters and background characters.  In most cases (except TEXT and
PICTURE in a report form), format strings are enclosed in quotes.

Format characters work as placeholders for displayed data and as "masks"
to limit the kind of data that can be entered.  There are separate format
characters numbers and strings (listed in the next two help screens).

Background characters are primarily decorative, added to make data easier
to read. They include all characters (including spaces) not listed in the
format character tables.  Background characters are displayed exactly
as they appear in the format clause, but are not stored as data.
(continues)

Next screen - <PgDn>, Previous Screen - <PgUp>, or Exit with <Esc>.....
          Enter topic
```

● Remember that the *SR-Info* database package includes a comprehensive Help system, describing commands and structures, with options and examples.